

RETI DI CALCOLATORI

PELLE

VOCABOLARIO

- SINCRONO / ASINCRONO
 - ↳ Non c'è prevista risposta
 - ↳ La richiesta del client prevede una risposta dal server
- BLOCCANTE / NON-BLOCCANTE
 - ↳ Il client fa richiesta al server e poi può fare tutt'altro
 - ↳ Il client si blocca finché non riceve la risposta dal server
- SIMMETRICO / ASIMMETRICO
 - ↳ C conosce S ma S non conosce C
 - ↳ C e S si conoscono
- STATICO / DINAMICO
 - ↳ Il legame viene fatto durante l'esecuzione → può cambiare → serrizio nomi
 - ↳ Il legame viene fatto prima dell'esecuzione → non può cambiare
- STATEFUL / STATELESS
 - ↳ Ogni messaggio è indipendente → no memoria → $\in <$
 - ↳ Ogni messaggio può dipendere da quelli precedenti → memoria → $\in >$
- GLOBALE / LOCALE
 - ↳ Interazioni circoscritte → forse scalabile
 - ↳ Non ci sono restrizioni alle interazioni → non scalabile
- TRASPARENTE / NON-TRASPARENTE
 - ↳ Non dipende dall'implementazione
- OVERHEAD → la quantità di risorse accessorie richieste oltre a quelle necessarie.
- SOCKET → sono i terminali locali o END-POINT di un canale di comunicazione bidirezionale

SEMANTICHE

- Best effort → fa il massimo possibile ma senza garantire controlli (errori, flusso, cong.)
MAY-BE
- AT - LEAST - ONCE → il messaggio deve arrivare, anche con ritrasmissioni, anche con dei duplicati
- AT - MOST - ONCE → si ha memoria di chi ha fatto richiesta e quando la richiede di nuovo si consegna il primo risultato
- EXACTLY - ONCE → insieme dei 2 precedenti, mira a garantire la consegna di solo 1 messaggio, prendendosi tutto il tempo necessario

MODELLO C/S

DEFAULT



Dove.

S → seq/parall.

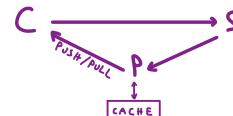
C → seq, con timeout

STRONG COUPLING

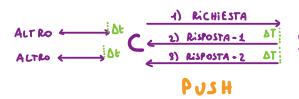
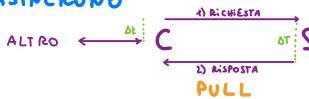
- SINCRONO + BLOCCANTE + ASIMMETRICO + DINAMICO

SINCRONO NON-BLOCCANTE → delegazione

C invia e poi può fare altro,
magari facendo ricevere risposte a proxy



ASINCRONO



LATO CLIENT



Ha sempre iniziativa

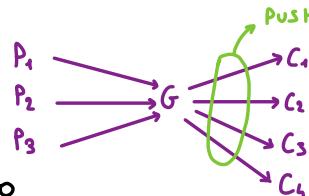


Ha iniziativa la prima volta, poi iniziativa al server

ALTRI MODELLI

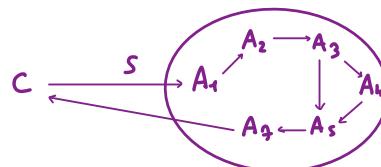
PUBLISH / SUBSCRIBE

- PRODUTTORI
+
GESTORE DI EVENTI
+
CONSUMATORI
- ASINCRONO e DISACCOPPIATO



AGENTI MULTIPLI

- SERVER = \sum AGENTE
+
CLIENT
↓
complezazione atomica
- PROTEZIONE e SCALABILITÀ



SISTEMA DI NOMI

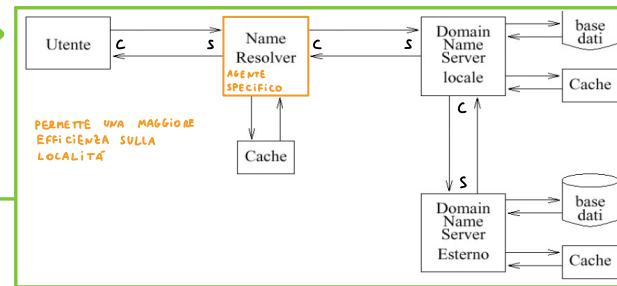
SISTEMA DI NOMI

NUOVE ENTITÀ INSERIBILI A RUN-TIME

- Permette BINDING DINAMICO nel distribuito
- Gestori PARTIZIONATI → ogni gestore ha solo la responsabilità di una stessa tabella
REPLICATI → ogni gestore collabora con gli altri sulla stessa tabella di nomi

DNS (Domain Names System)

- HOST $\xleftarrow[\text{DNS}]{}$ nome logico $\xrightarrow[\text{IP}]{}$ nome fisico
 $\xrightarrow[\text{max 127} \rightarrow \text{per vincoli fisici}]{}$ PAROLA ≤ 63 char
 $\xrightarrow[\text{TOT} \leq 255 \text{ char}]{}$
- STRUTTURA AD ALBERO → vari livelli → principio di località dei servizi
 - ↳ varie zone → ragioni geografiche e/o organizzative
 - ↳ responsabilità specifiche
 - ↳ suddivise per delegazione
- REQUISITI → affidabilità, efficienza e località



NECESSITÀ DI REPLICATIONE

ZONA

PRIMARY MASTER

REPLICAZIONE

SECONDARY MASTER

→ non rispondono alle query, richiedono aggiornam.
al primary (PULL) e lo sostituiscono nei guasti

- QUERY → ricorsiva → catena di request/reply tra DNS Servers → RISP: OK/ERRORE
- ITERATIVA → 1 DNS Server fa sequenza di request/reply → RISP: OK/SUBB.
anche inverse

DIRECTORY X.500

- Sistema di nomi in grado di catalogare qualunque informazione e renderla sempre disponibile con un minimo di qualità.
- È composta da agenti che memorizzano parti della directory e che insieme permettono il servizio.

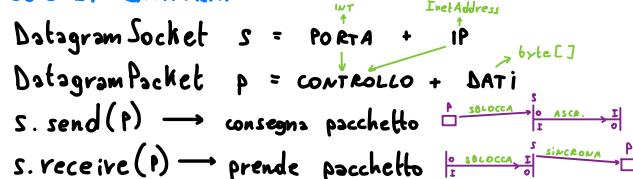
SOCKET JAVA

(pkg: java.net)

SOCKET

- con connessione → TCP → qualità >> e € >>
- senza connessione → UDP → qualità << e € <<
- Identifica processo con nome globale = IP + PORTA (no PID!)

SOCKET DATAGRAM



SOCKET MULTICAST

- MulticastSocket s = PORTA
 s.join(group(IP)) → iscrizione
 s.leave(group(IP)) → disiscrizione
- tra queste 2 operazioni si usano le stesse API di DatagramSocket → il multicast è per forza senza connessione
- OPZIONI → setSoTimeout
 setSendBuffer
 setReceiveBuffer } PER MODIFICARE COMPORTAMENTO DEFAULT
- PS: IP è classe D → [224.0.0.0 - 239.255.255.255]

SOCKET STREAM

- client = PORTA + IP oppure (PORTA-LOC, IP-LOC, PORTA-REM, PORTA-REM)
 server = PORTA (+ CODA LEN)
 Socket conn-client = server.accept() → sincrona bloccante
- I/O → byte in arrivo/partenza vengono bufferizzati da adapter per formato



client.close() → chiusura immediata ma brusca → perdita dati in transito

client.shutdownOutput()

< attendo che pari chiude output > → chiusura di output, gestione dati in transito e chiusura di input
 client.shutdownInput()

• OPZIONI → settare linger → tempo da aspettare prima di buttare dati OUTPUT dopo close

SOCKET C

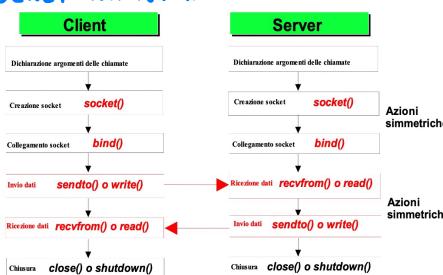
UNIX

- SEGNALI → sincronizzazione
- + FILE → comunicazione
- file descriptor → socket descriptor → uso primitive
- `sockaddr` → indirizzo generico socket
- + `sockaddr_in` → indirizzi di internet
- `hostent*` `gethostbyname (char* name)` → servizio di nomi
- `servent*` `getserverbyname (char* name, char* proto)` → rilevare porta di un servizio

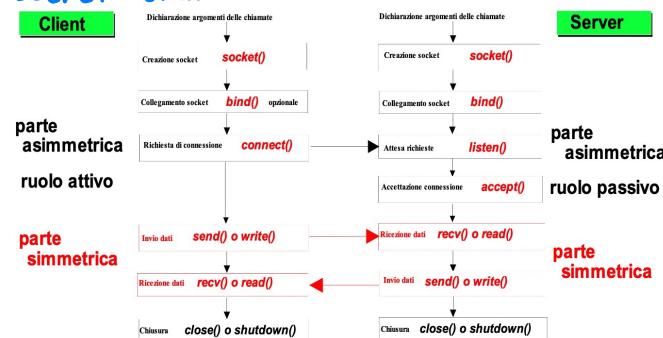
PRIMITIVE

`int socket (int dominio, int tipo, int protocollo)` → CREAZIONE LOCALE SOCKET
`int bind (int s, sockaddr* indirizzo, int lunghezza)` → CREAZIONE HALF-ASSOCIATION
`int connect (int sd, sockaddr_in* addr, int addrlen)` → VIENE FATTA RICHIESTA SINCRONA DI CONNESSIONE AL SERVER
 LUNGHEZZA CODA
 TERMINA QUANDO LA RICHIESTA È ANCHE SOLO ACCODATA o ERRORE
 INVOCÀ ANCHE LA BIND, SE NON ANCORA FATTA
`int listen (int sd, int backlog)` → ASCOLTA RICHIESTE DI CONNESSIONE E LE METTE IN CODA PER `accept()`
`int accept (int ls, sockaddr_in* addr, int* addrlen)` → ACCETTA RICHIESTA E CREA CONNESSIONE
`int read (int sd, char* buff, int length)` → LETTURA DI UN BUFFER DI BYTE → SEMPRE SINCRONA CON DRIVER TCP
`int write (int sd, char* buff, int length)` → SCRITTURA DI UN BUFFER DI BYTE → NON SINCRONA CON LETTURA (LIV. APPL.)
`int close (int sd)` → SCOLLEGA SUBITO PROCESSO DA SOCKET → SO → RILASCIO RISORSE LOCALI
 DRIVER → Δt?? → RILASCIO RISORSE REMOTE
 OUTPUT → SPEDITO + EOF
 INPUT → BUTTATO
`int shutdown (int sd, int how)` → CHIUSA DOLCEMENTE CONNESSIONE
 → 0 → chiude IN
 → 1 → chiude OUT
 → 2 → chiude IN/OUT

SOCKET DATAGRAM



SOCKET STREAM



ATTESA MULTIPLO

- Processo attende su più socket differenti contemporaneamente per poi avere comportamenti diversi → OPERAZIONE BLOCCANTE IN UNA SOCKET IMPEDIREBBE AZIONI SULL'ALTRA!!

SOLUZIONE

int select (size_t max_eventi, int* mask_read, int* mask_write, int* mask_except, timeval* timeout_max)

AZIONE SOSPENSIVA E BLOCCANTE

La select aspetta fino alla scadenza del timeout.

Restituisce il numero di eventi accorsi e segnalati nelle maschere

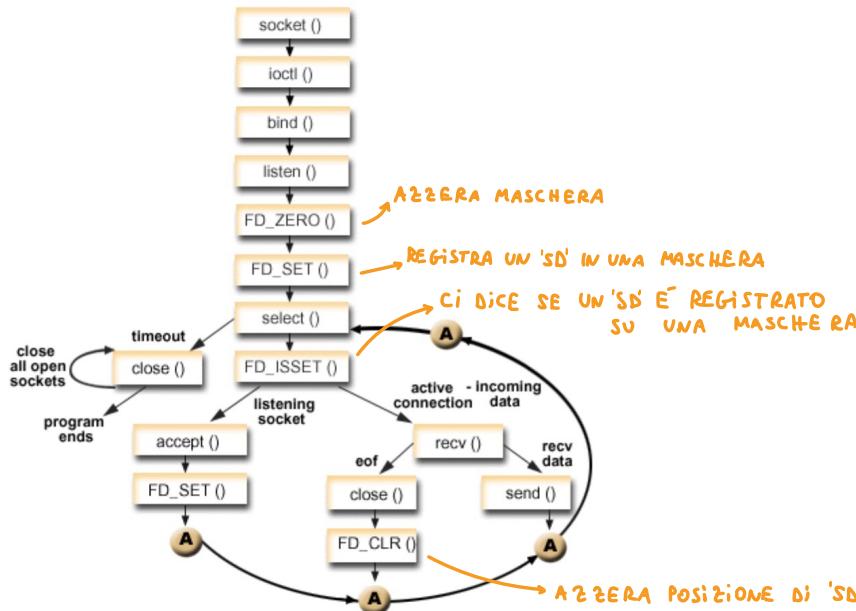
VALORE FD/SD	0	1	2	3	4	5	6	...
MASK_READ	0	0	0	1	0	1	1	..
MASK_WRITE	0	0	0	0	1	1	1	..
MASK_EXCEPT	0	0	0	0	0	1	0	..

IMPIEGATI PER STANDARD ←

VALORE FD/SD	0	1	2	3	4	5	6	...
MASK_READ	0	0	0	1	0	0	1	..
MASK_WRITE	0	0	0	0	1	0	1	..
MASK_EXCEPT	0	0	0	0	0	0	0	..

→ accept HA CREATO NUOVA SOCKET CON SD = 5

close(s)

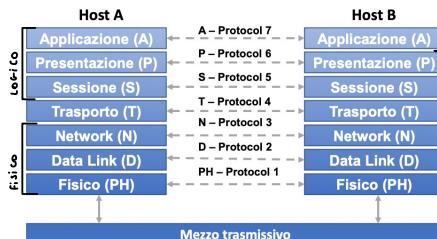


OSI (Open System Interconnection)

COS'E' PROGETTO → NON IMPLEMENTATO

- E' uno standard di comunicazione tra sistemi aperti con l'obiettivo dell'interoperabilità tra sistemi diversi.

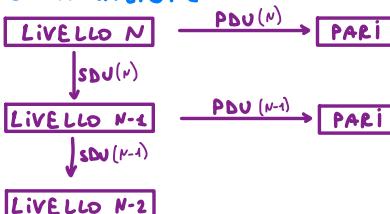
- Presenta una struttura a livelli



SAP → interfaccia tra livelli → offre servizio a livello sopra
 SAP address → nome unico → per comunicare con remoti dovremo sapere tutti i nomi
 MA → uso SAP-RETE → identificativo univoco e abbreviato
 → mai sotto al livello 3



COMUNICAZIONE



ASSEMBLAGGIO E DISASSEMBLAGGIO
 Assemblaggio a header:
 $PDU(N-1) = \text{header}(N-1) + SDU(N)$ → COMPOSIZIONE MSG
 $SDU(N-1) = PDU(N-1)$ → PASSIAMO SOTTO PER ALTRE COMPOSIZIONI

CONNESSIONE

CONNECTIONLESS → NO ORDINE, COSTI LIMITATI, SCARSE GARANZIE, NO QoS, NO STORIA, NO NEGOZIAZIONE

CONNECTION-ORIENTED → SI ORDINE, COSTI ELEVATI, ALTE GARANZIE, SI QoS, SI STORIA, SI NEGOZIAZIONE

PRIMITIVE

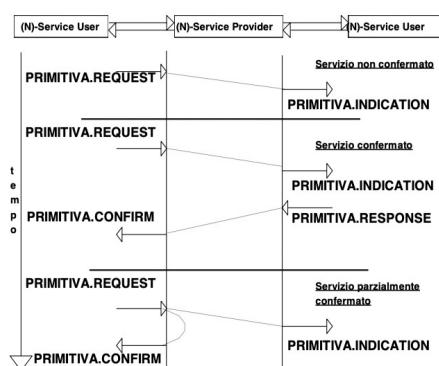
Operazioni atomiche usate dai livelli per implementare le loro funzionalità

Sono CONNECT → DATA → DISCONNECT



RE REQUEST → INDICATION → RESPONSE → CONFIRM

SERVICE USER CHIEDE SERVIZIO
 SERVICE PROVIDER INDICA RICHIESTA
 SERVICE USER SPECIFICA RISPOSTA
 SERVICE PROVIDER SEGNALETTA RISPOSTA ALLA RICHIESTA



LIVELLI BASSI (FISICO, DATA - LINK, NETWORK, TRASPORTO)

Insieme forniscono un meccanismo trasparente per avere un trasporto con:
CONTROLLO ERRORE, CONTROLLO FLUSSO, NAMING, ROUTING

NETWORK

Tiene conto dei nodi intermedi tra 2 pari.

Realizza ROUTING + CONTROLLO FLUSSO + CONTROLLO CONGESTIONE

TRASPORTO

Spezza il dato per il NETWORK e dall'altra parte lo ricomponne per la SESSIONE.

Lavora in modo end-to-end per separare FISICA e LOGICA.

primitiva	tipo di servizio	parametri di servizio
T-CONNECT	servizio confermato	indirizzo del chiamante e del chiamato, opzione per l'uso di dati privilegiati, qualità di servizio e dati d'utente.
T-DATA	servizio non confermato	dati di utente
T-EXPEDITED-DATA	servizio non confermato	dati di utente
T-DISCONNECT	servizio non confermato	ragione della terminazione, dati d'utente

Dati EXPEDIT → CONTROLLO DI FLUSSO SEPARATO PER AVERE L'INVIO DEI MESSAGGI DI CONTROLLO ANCHE SE I MESSAGGI NORMALI SONO BLOCCATI

SESSIONE

BIDIREZIONALE, MULTIPLE, STRUTTURATO, GARANTE

Considera e determina i meccanismi per il dialogo tra entità ≠, contando la possibilità di una comunicazione varia ed eterogenea.

Realizza SINCRONIZZ. + APRI/CHIUDI CONN + MODALITÀ DIALOGO + TRASF. DATI + AUTO 22.



Fornisce supporto al recovery dello stato tramite

- 1) ABBANDONO → RESET DELLA COMUNICAZIONE
- 2) RIPRISTINO → COMUNICAZIONE PORTATA AL CHECKPOINT MAGGIORE PRECEDENTE
- 3) RIPRISTINO DIRETTO DALL'UTENTE → STATO ARBITRARIO SENZA CONTROLLI

PRESENTAZIONE

Si preoccupa di uniformare i dati dei due pari che possono essere omogenei o no.

1) DATI OMOGENEI → SITUAZIONE IDEALE, NON MOLTO INTERESSANTE

2) DATI NON OMOGENEI ↗ OGNI MODO PUÒ CONVERTIRE DATI → ELEVATA PERFORMANCE

↘ CONCORDARE FORMATO COMUNE → MEMO FUNZIONI DA IMPLEMENTARE

ETEROGENEITÀ ↗ COME DIRLO → ACCORDO CONCRETO → PERMETTE RIDOMANDA

COSA DICE → ACCORDO ASTRATTO

NEGOZIAZIONE (BIDDING tra Sender e Receivers)

S: manda broadcast (ANNOUNCE)

R: fanno offerte (BIDDING)

S: sceglie offerta (AWARD)

R: riceve scelta (CONTRACT) → OGNIUNO CAPISCE SE È STATO SCELTO O meno

S: instaura accordo

• Fornisce 2 linguaggi

SCRIVE FUNZ. DI ACCORDO FRA ENTITÀ NON ACCORDATE STATICAMENTE → CASI DINAMICI

1) ASN.1 → LINGUAGGIO ASTRATTO, DESCRIVE UNIVOCAMENTE CONTESTO, SOGGETTO, SEMANTICA

DOPPO SI PASSA
COMUNQUE A

2) BER → LINGUAGGIO CONCRETO, STABILISCE REGOLE DI DECODIFICA DEI DATI → TUTTI SANNO COME DECODIFICARE

APPLICAZIONE

Si interfaccia con l'utente e, attraverso l'estrazione, ha il compito di nascondere la complessità dei livelli sottostanti.

MODELLO AD OGGETTI → USA TEMPLATE PER DEFINIRE OGGETTI → ESPRESSI ATRAVERSO PACKAGE
EREDITÀ STATICA

Offre servizi come X.500 → CLASSIFICA E COLLOCA OGNI ENTITÀ DI INTERESSE IN SISTEMA GERARCHICO → ALBERO

↘ 24/7
MOLTO ACCESSIBILE

RETI E ROUTING

RETI

Sono il mezzo di interconnessione punto a punto o di gruppo che coordinano molte entità.

Si giudicano in base a: LATENZA, BANDA, CONNETTIVITÀ, COSTO COMPONENTI, AFFIDABILITÀ, FUNZIONALITÀ

Possono essere **DIRETTE/STATICHE** o **INDIRETTE/DINAMICHE** o **COMPLETE** → tutti connessi con tutti
NON VARIABILI, SCELTE A PRIORI ↴ VARIABILI, COLLEGATE A SWITCH → CONDIVISIONE RISORSE

LAN

Sono caratterizzate da ALTA VELOCITÀ, AMPIA BANDA, POCHI ERRORI, BROADCAST FACILITATI

Tendono alla ridondanza → alte prestazioni

INTERCONNESSIONI

RIPETITORE + **BRIDGE** + **ROUTER** + **PROTOCOL CONVERT**

↓
FISICO

↓
DATA-LINK

↓
NETWORK

↓
TRASPORTO

BRIDGE

Mantiene il traffico di una parte e non fa entrare il traffico non necessario.

PRO	CONTRO
<ul style="list-style-type: none"> SEPARAZIONE RETI BUFFERRIZZA I FRAME GESTISCE VARI ACCESSI 	<ul style="list-style-type: none"> BUFFERZAZIONE LIMITATA RTARDO BUFFERZAZIONE FRAME TRASFORMATO

Possono essere **MULTIPORTA** o **TRASPARENTE**

COLLEGANO SEGMENTI DI RETE DIVERSI

OPERANO IN MODO INVISIBILE AGLI UTENTI

PRIMA PASSA TUTTO E Poi FILTRA IN BASE AI FEEDBACK

Hanno tabella di FORWARDING (MAC + PORTA) che si autocompila in fase di **LEARNING**.

Se ci sono più bridge → **SPANNING TREE** → ROUTER SI PARLANO E TROVANO I COSTI PIÙ BASSI DI COLLEGAMENTO E COSTAVISCONO UN ALBERO UNICO DI INTERCONNESSIONI

ROUTING

Dove essere CORRETTO + SEMPLICE + ROBUSTO + OTTIMALE + GIUSTO

Può essere **LOCALE/GLOBALE** + **STATICO/DINAMICO** + **ADATTIVO/NON-ADATTIVO**

IN BASE ALLA VISIBILITÀ

IN BASE A QUANTO SONO DETERMINATI I CAMMINI

SE SFRUTTA O MENO LE RISORSE DINAMICAMENTE

Il router può avere problemi di

CONTESTIONE

BUFFER DEI ROUTER SONO PIEMI
PREVE DIAMO MIN MAX. MSF. CIRCOLANTI

MESSAGGI DI INDICAZIONE
AL MITTENTE
SCARTIAMO I MESSAGGI SUCC. A UNO ATTESO

DEADLOCK

BLOCCO IMPOSSIBILE X TOTALI OUT.
AVOIDANCE
BUFF. NUMERATI E ORDINATI
RECOVERY
TRATTIAMO IL PROBLEMA DOPO

PREVENTION
BUFF. SERVITI IN CASO DI SATURAZIONE

LIVE LOCK

MESSAGGI PERDUTI IN CICLI
PRIORI & NO CICLI
POSTERIORI & POST-TO-LIVE

ALGORITMI DI ROUTING

GLOBALI → propagazione globale di info

→ **SHORTEST PATH FIRST**

OGNI NODO SI COSTRUISCE GRAFO COMPLETO DI INTERCONNESSIONI, LE PESA E NE MISURA LE DISTANZE PER Poi TROVARE IL PERCORSO MIGLIORE AD OGNI MESSAGGIO

ITERATIVO → PARTE DA ADIACENTI, poi A 2 salti, poi A 3 salti ...

→ **SPANNING TREE**

→ **LINK STATE** → OGNI NODO MANTIENE TUTTO IL GRAFO E LIMITA PROPAGAZIONI

CONTROLLO CONNESSIONE

INVIO PERIODICO DI MESSAGGI PER CONTROLLARE CORRETTEZZA

DECESSO TRAMITE

MESSAGGI SPECIFICI

BROADCAST/FLOODING DEL MSG

PRO

- CONTROLLO VIZIATO
- VARIAZIONI PROPAGATE VELOC.
- CAMMINI POSSONO ESSERE X
- MULTIPATH
- CONOSCENZA COMPLETA CAMMINI

CONTRO

- COSTO MANUTENZIONE TOPOLOGIA
- AZIONI COSTOSE PER VARIAZIONE BROADCAST

→ **DISTANCE VECTOR** → ALGORITMO DINAMICO IN CUI OGNI ROUTER MANTIENE UNA TABELLA (DISTANZA IN PASSI; PRIMO PASSO D'USCITA)

COUNTING TO INFINITY ↴

NON SI TIENE TRACCIA DI CHI FORNISCE LA DISTANZA DA UN NODO SI CREA LOOP INFINITO

es: A-B-C
Se A va giù, B non riceve il messaggio ma C, che ha imparato A da B, dirà di poter raggiungere in 2 passi

PROPAGAZIONE MOLTO LENTA → ESPONENZIALE AL NUMERO DI MODI

SPLIT HORIZON

QUANDO VOLTISSIMO LA MISTRA TABELLA DI ROUTER A UN ROUTER VIZIO, OMETTIAMO I PERCOLSI CHE ABBIANO APPRESO DA LUI

SPLIT HORIZON CON POISON REVERSE + BROADCAST

INVIAMO METÀ INFINITA RICARICO AL RAGGRUPPAMENTO DI UN NODO AL NODO CHE LO HA INSERITO, ESCLUDENDOLO DAL ROUTING

AVVISIAMO TUTTI DI ERRORE

HOLD DOWN

QUANDO APPRENDEMOSI UN ERRORE SU UNA ROTTA, QUESTA VIENE ISOLATA E NON CI CONDUCONO AGGIORNAMENTI

ISOLATI → no coordinamento → no propagazione

→ **RANDOM** → IL MESSAGGIO VIENE SMISTATO IN UNA CODA CASUALE TRAMME L'INPUT

→ **PATATA BOLLENTE** → IL MESSAGGIO VIENE SMISTATO SULLA CODA PIÙ SCARICA

→ **FLOODING** → IL MESSAGGIO VIENE SMISTATO IN TUTTE LE CODE DI USCITA DEL NODO

DATAGRAMMI IP

È l'unità base di informazione che viaggia in Internet.

LAN) IP = HOST ID (21 bit) + NET ID (8 bit)

VERS	HLEN	SRV	TP	TOTAL LENGTH		
IDENTIFICATION				FLAGS	FRAGMENT	OFFSET
TIME TO LIVE	PROTCL	HEADER CHECKSUM				
SOURCE IP ADDRESS						
DESTINATION IP ADDRESS						
IP OPTIONS (if any)			PADDING			
DATA						
...						

0 3 SeRVice Type
PRECEDENCE D T R C UNUSED

FLAGS
Do not fragment More fragments UNUSED

C → COSTO MINIMO
R → MASSIMA AFFIDABILITÀ
T → MASSIMO THROUGHTPUT → MUL. INFO TRASMISSIONIBILI
D → MINIMO RITARDO

1 → VERSIONE + LUNGHEZZA HEADER + TIPO DI SERVIZIO + LUNGHEZZA totale
2 → IDENTIFICATORE + FLAGS + FRAGMENT OFFSET
3 → TEMPO DI VITA + PROTOCOLLO + CHECKSUM
4 → IP MITTENTE
5 → IP DESTINATARIO
(6,16) → CONNECTIONLESS + UNRELIABLE

IP è: no QoS, best effort, è «

Durante la trasmissione, ogni router può frammentarlo in base al proprio MTU ma sarà ricomposto solo dal destinatario

TCP/IP

ARP

- Protocollo che fornisce traduzione da IP a MAC.

Per completare la propria tabella, a ogni richiesta, se conosce IP lo risolve altrimenti lo manda in broadcast aspettando che macchina corrispondente gli invii il MAC.

RARP

- Protocollo che fornisce traduzione da MAC a IP. → IN CASO DI PICCOLE MACCHINE SENZA DISCO

Non ha modo di completare dinamicamente la propria tabella, quindi se conosce IP lo restituisce, altrimenti dà risposta negativa.

Per questo motivo ci sono molti server RARP anche in una stessa rete e questi possono rispondere:

DINAMICAMENTE → PRIMA IL PRIMARIO, Poi segue la gerarchia

STATICALMENTE → RISPOSTANO TUTTI MA CON RITARDI CASUALI → PER EVITARE RISPOSTE SIMULTANEE

DHCP

- Protocollo per l'attribuzione dinamica di indirizzi IP per ottenere una configurazione dinamica che risparmia su IP.

Si basa su asta con C/S:

- C invia DHCPDISCOVER
- I vari S ricevono richiesta e se vogliono fanno offerta con DHCPOFFER
- C aspetta varie offerte, sceglie una e fa broadcast di scelta con DHCPREQUEST
- S che offre conferma con DHCPACK, gli altri capiscono di non essere stati presi
- In fine avremo DHCPLEASE e/o DHCPRELEASE per indirizzi inutilizzati

Il contratto è a tempo → FAVORISCE RICAMBIO DI INDIRIZZI IP

Usa UDP con S:67 e C:68

NAT

- Protocollo usato per traslare indirizzi IP privati in IP globali, tramite dei router mantiene una tabella di traduzione.

Dovrebbe istruire i DNS per fare delle traslazioni articolate per consistenza.

Usando il NAT A PORTE abbiamo 1 IP esterno per tutti gli IP interni

ICMP

- Protocollo che si occupa della gestione e del controllo degli IP
I messaggi ICMP sono incapsulati dei datagrammi IP e segue il normale routing.
Tramite ping e traceroute possiamo verificare la raggiungibilità di un sistema e verificare i cambi di percorso

UDP

- Protocollo di trasporto MAY-BE, basso costo, poco overhead, unreliable, con duplicati, disordine e perdite di messaggi.
Infatti i suoi datagrammi sono composti da MITTENTE + DEST + CHECKSUM + DATI

CONTINUOUS REQUEST

Non si attende in modo sincrono l'ACK ma si mandano messaggi fino a saturare la propria finestra per poi bloccarsi.

In caso di errore si fa:

- 1) **SELECTIVE RETRANSMISSION** → C'È L'ATTESA DELL'ESITO DEI MESSAGGI E RITRASMISSIONE SOLO DI QUELLI PERSI
- 2) **GO BACK N** → RITRASMISSIONE DI N MESSAGGI SE UNO HA DEI PROBLEMI → BUFFER VIENE SVUOTATO

TCP

- Protocollo per la trasmissione AFFIDABILE, ORDINATA, AT-MOST-ONCE, senza impiegare i nodi intermedi

Nell'header abbiamo molto overhead

0	4	10	16	24	31
SOURCE PORT	DESTINATION PORT				
SEQUENCE NUMBER					
ACKNOWLEDGEMENT NUMBER					
HLEN	RSRVD	CODE BIT	WINDOW		
CHECKSUM		URGENT POINTER			
OPTIONS (IF ANY)		PADDING			
DATA					
...					

URG → DATO URGENTE
ACK → CONFIRMA
PUSH → INVIO IMMEDIATO
RST → RESET COMM.
SYN → INIZIO COMM.
FIN → FINE COMM.

DETERMINATA DAL RICEVENTE E RICOMUNICATA AD Ogni INVIO

Nella comunicazione viene usato CONTINUOUS REQUEST, tramite SLIDING WINDOW che viene saturata in attesa di ACK per continuare o rinviare il flusso.

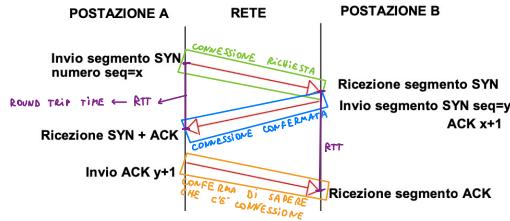
Se non c'è ricezione segmento si usa GO BACK N → FINCHE' PUÒ LI SALVA OUVIAMENTE SCARICA SEGMENTI SUCCESSIVI, ASPETTAVO CHE MITTENTE FISI INVII QUELLO MANCANTE

Vengono usati gli ACK CUMULATIVI che rinvia anche dati già ricevuti

FASI OPERATIVE DI TCP

FASE INIZIALE

Viene iniziata la comunicazione mediante il THREE WAY HANDSHAKE

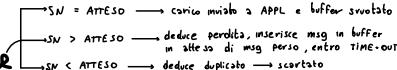


- X e Y sono casuali e $\neq \emptyset$
- SI CONCORDANO ANCHE SU
 - MSS \rightarrow dimensione massima del blocco di dati
 - DIMENSIONE FINESTRA DI RICEZIONE
 - COORDINAMENTO OROLOGI \rightarrow si decidono i timer \rightarrow almeno RTT

FASE COMUNICAZIONE

Qui troviamo il continuous REQUEST con GO BACK N ad ACK CUMULATIVI per le ritrasmissioni.

Per ordine e no duplicati abbiamo il SEQUENCE NUMBER



• CONTROLLO FLUSSO \rightarrow EVITARE MITTENTE INVII TROPPI DATI PER NON CONGESTIONARE IL PARI
 \downarrow MSS \rightarrow LUNGHEZZA OTTIMIZZATA PER BUFFER \rightarrow Nagle: se ci sono messaggi piccoli li metto insieme fino a raggiungere MSS
 FINESTRA \rightarrow STATO DI MEMORIA DEL RICEVITORE \rightarrow RWND

• CONTROLLO CONGESTIONE \rightarrow EVITARE MITTENTE INVII TROPPI DATI PER NON CONGESTIONARE LA RETE \rightarrow BUFFER DEI ROUTER PIENI
 \downarrow RECOVERY ALLO SCATTARE DI TIMER \rightarrow SLOW START \downarrow VIENE USATO SIA ALL'INIZIO CHE A REGIME

Subito dopo aver stabilito la connessione, la comunicazione parte in SLOW START che evita congestione tramite: RWND + CWND + SSTHRESH

RECEIVING WINDOW \downarrow CONGESTION WINDOW \downarrow SOGLIA DI SLOW START
 PARTE DA MSS FINO A RWND

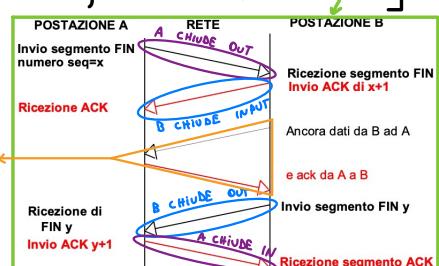
ALG. SLOW START : se $CWND_i < SSTHRESH \rightarrow CWND_f = 2 \cdot CWND_i$
 RISCONTRATA CONGESTIONE TRAMITE I TIME-OUT \downarrow se $CWND_i > SSTHRESH$ AND $CWND_i < RWND \rightarrow CWND_f = CWND_i + 1$
 \downarrow $CWND_f = 1$, $SSTHRESH = CWND_i / 2$

Trucchi di TCP \leftarrow TIME-OUT DINAMICO \rightarrow TIME-OUT RADDOPPIA CON RITRASMISSIONE
 EXPONENTIAL BACKOFF \rightarrow SILLY WINDOW \rightarrow RWND = \emptyset FINO A MSS/2 \rightarrow MENO SEGMENTI CORTI
 LONG FAT PIPES \rightarrow FUNGO DI AVERE BUFF GRANDI \rightarrow PIPE SEMPRE PIENE

FASE FINALE

Viene terminata la connessione [close() = 3-WAY ; shutdown() = 4-WAY]

UGUALE AD APERTURA MA MANDIAMO FIN INVECE DI SYN



RMI (Remote Method Invocation)

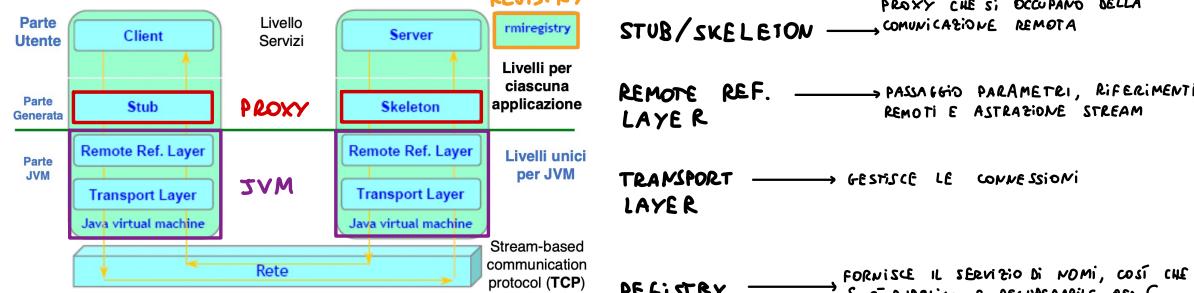
COSA SONO

Metodo per offrire la possibilità di eseguire metodi remoti in Java.

JAVA HA SEMANTICA X RIFERIMENTO → OGNI VARIABILE CHE RIFERISCE UN'ISTANZA È UN RIFERIMENTO NELLA MEMORIA
↳ REFERENCE COUNTING → GARBAGE COLLECTOR

- In locale usiamo una VARIABILE INTERFACCIA che contiene riferimento a PROXY che permette di controllare/preparare il passaggio del messaggio.
- RMI lavora con TCP con semantica SINCRONA BLOCCANTE

ARCHITETTURA



OGGETTI

Usiamo INTERFAZIE + CLASSI + METODI tutti remoti

IMPLEMENTATIONS → EXTENDS → throws → INVOCAZIONE REMOTA
java.rmi.Remote → java.rmi.UnicastRemoteObject → java.rmi.RemoteException → NON È COMPLETAMENTE TRASPARENTE

PASSI DI UTILIZZO

1) DEFINIRE SERVER REMOTO (INTERFAZIA + IMPLEMENTAZIONE)

```
javac ServiceInterface.java ServiceRMIServer.java
```

2) GENERARE STUB E SKEL

NON SI INSERISCE NULLA NEL CLIENT PERCHÉ LO STUB È LATO CLIENT, MA
rmic -vcompat ServiceRMIServer → RAPPRESENTA IL SERVER NEL CLIENT
↳ INFATTI BASTA SOLO SERVER

3) AVVIARE IL REGISTRY E REGISTRARE IL SERVIZIO

rmiregistry ; java ServiceRMIServer → rmiregistry è di default su 1099 a meno che non specifichiamo noi
↳ DEVE FARLE BIND (IPregistry:porta/servizio) IN MAIN

4) CREARE CLIENTE CHE DOVRÀ FARE LOOKUP AL REGISTRY

```
javac ServiceRMIClient.java ; java ServiceRMIClient  
↳ DEVE FARLE NAMING.lookup (IPregistry:porta/servizio)
```

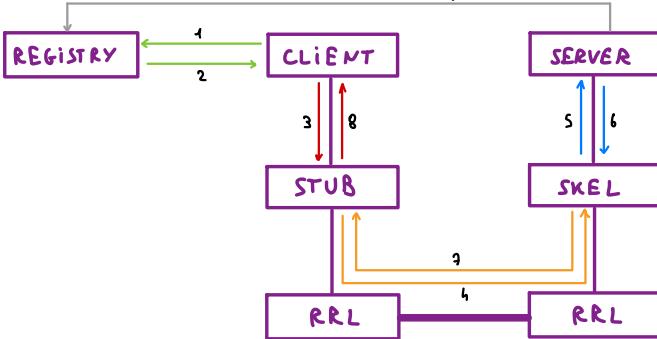
RMI REGISTRY

È in esecuzione su una JVM a parte e nota al cliente così che possa fornirgli la referenza al servizio in base al nome dello stesso.

Il nome del servizio è **REGISTRY + NOME SERVIZIO** → `://IP_Host:porta/nomeServizio`

**LOCALE E NON
GLOBALE**
IN-THE-SMALL

COMUNICAZIONE STUB - SKEL



- 0) SERVER REGISTRA SERVIZIO SU REGISTRY
- 1) CLIENT INTERROGA REGISTRY "SERVIZIO?"
- 2) REGISTRY FORNISCE IP:PORTA X SERVER
- 3) CLIENT FA RICHIESTA A STUB X SERVIZIO
- 4) STUB SERIALIZZA E PASSA A SKEL
- 5) SKEL DESERIALIZZA E INVOCIA SERVIZIO
- 6) SERVER ESEGUE SERVIZIO E PASSA RESULT A SKEL
- 7) SKEL SERIALIZZA RISULTATO E PASSA A STUB
- 8) STUB DESERIALIZZA E PASSA A CLIENT

PASSAGGIO PARAMETRI

Primitivi → VALORE

Oggetti locali → SERIALIZABLE

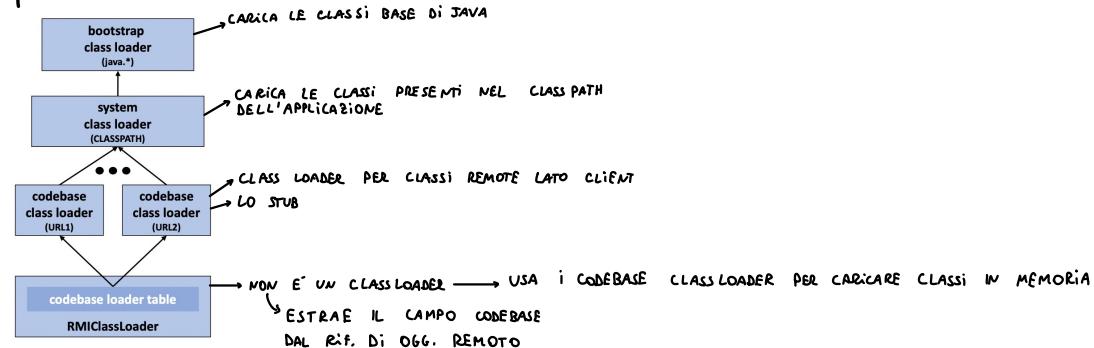
Oggetti remoti → REMOTE

CONCORRENZA E COMUNICAZIONE

I server sono in parallelo, grazie a JVM che ha processo in ascolto per generazione thread, e quindi il livello applicativo deve essere **thread-safe** SYNCHRONIZED

CLASS LOADER

Entità capace di risolvere problemi di caricamento delle classi dinamicamente e di trovarle quando servono.



RPC (Remote Procedure Call)

PROPRIETÀ

- TRASPARENZA
- TYPE CHECKING
- CONTROLLO CONCORRENZA/ECCEZIONI
- BINDING DISTRIBUITO
- TRATTAMENTO DEGLI ORFANI → PROCESSO SERVER CHE NON RIESCE A FORNIRE IL RISULTATO

TOLLE RANZA AI GUASTI

Si ha l'obiettivo di mascherare malfunzionamenti

LATO CLIENT → ERRORE SERVER

MAY-BE → ATTESA

AT-LEAST-ONCE → ATTESA + RITRASMISSIONE

AT-MOST-ONCE → MEMORIA AZIONI

EXACTLY-ONCE → AZIONE VIENE FATTA

LATO SERVER → ERRORE CLIENT → PROCESSI ORFANI

STERMINIO → ORFANI DISTROTTI

TERMINAZIONE A TEMPO → AZIONE CON TIME-OUT

REINCARNAZIONE → TEMPO DIVISO IN EPOCHE PER DETERMINARE AZIONI OBSOLETE

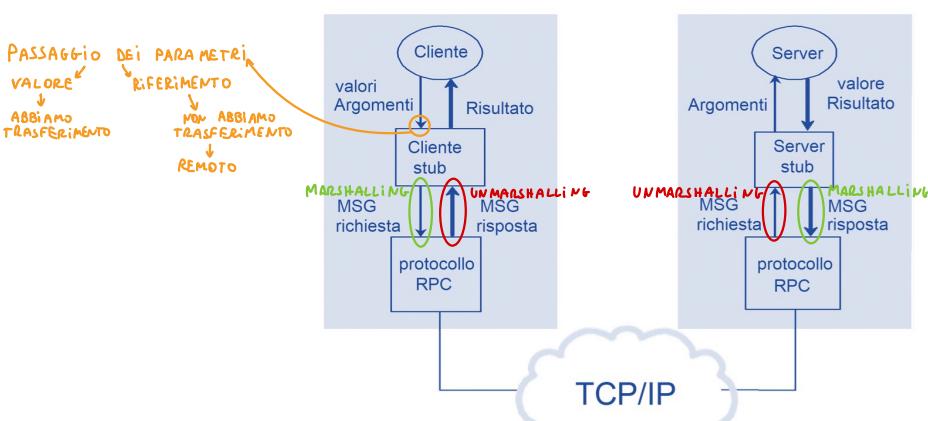
ONC (Open Network Computing)

Sun propone callrpc (name-nodo-remoto, id-procedura, specifica-trasformazione)

MODELLO NON TRASPARENTE → BISOGNA FARE CHIAMATA E INSERIRE I PARAMETRI E GESTIONE ERRORE → DIPENDENZA DA TECNOLOGIA

NCA (Network Computing Architecture)

Cerca di introdurre trasparenza tramite 2 stub → si percepiscono come chiamate locali

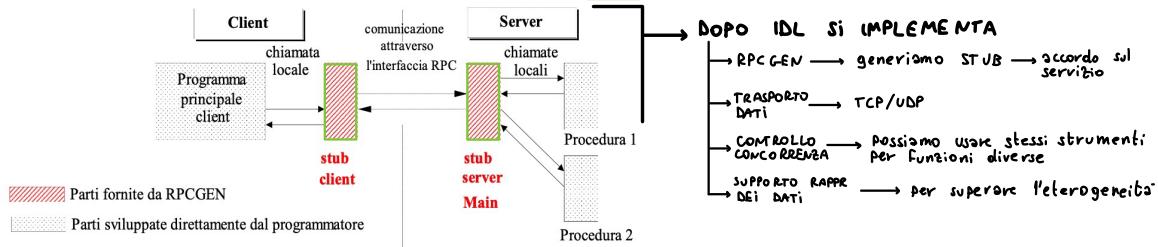


IDL (Interface Definition Language)

Sono linguaggi per descrivere operazioni remote → IDENTIFICAZIONE UNICA SERVIZIO

DEFINIZIONE ASTRATTA DEI DATI

Per Sun c'è XDR dove utente descrive la logica e poi la traduce in codice tramite RPCGEN → si passa da IDL a STUB



RPC BINDING

STATICO e/o DINAMICO

C/S

SOLUZIONE COSTOSA SE FATTA AD OGNI CHIAMATA

se ci sono chiamate allo stesso server, memorizziamo e usiamo sempre lo stesso legame

SERVIZIO + INDIRIZZAMENTO

STATICO

client specifica a chi vuole connettersi con NAMING
ID NUMERICO x INTERFACCIA SERVIZIO

DINAMICO

client è già collegato al server tramite ADDRESSING

ESPlicito

richiesta C/S fatta in broadcast

IMPLICITO

richiesta C/S fatta tramite server di nomi

lookup(servizio, versione, &server)
register(servizio, versione, servitore)
unregister(servizio, versione, servitore)

RPC ASINCRONE

Sono necessarie perché non vogliamo bloccare il client per aspettare una risposta dal server, 2 opzioni

BASSA LATENZA → trascurano il risultato
ALTO THROUGHPUT → più richieste in una ms con risultato

REALMENTE ASINCRONE ???

RPC VS RMI

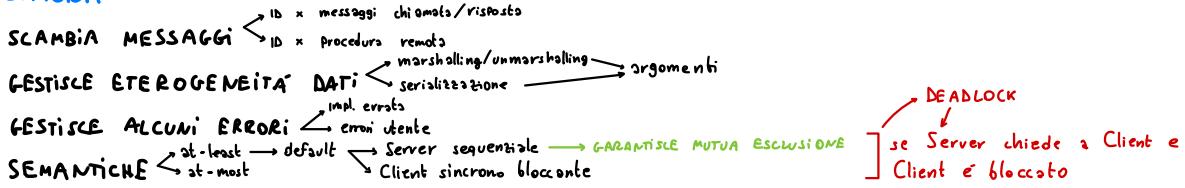
SEQUENZIALE A DEFAULT

PARALLELO

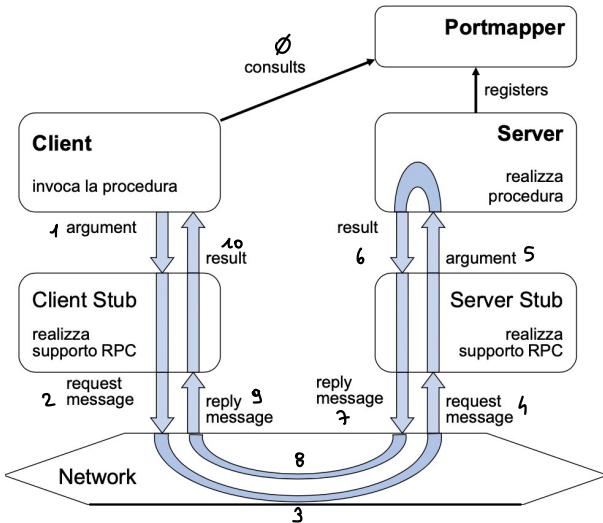
	SUN RPC	RMI
Entità da richiedere:	solo operazioni e funzioni	metodi di oggetti via interfacce
Semantica:	at-most-once, at-least-once	at-most-once (TCP)
Comunicazione:	sincrona e asincrona	solo sincrone
Durata - Errori:	timeout	trattamento di casi di errori
Ricerca Server:	portmapper sul server	registry nel sistema unico centrale
Presentazione dei dati:	XDR e RPCGEN (XDR)	generazione stub e dello skeleton
Passaggio parametri:	per valore, strutture complesse sono linearizzate e ricostruite al server	per valore di default, per riferimento gli oggetti con interfacce remotizzabili

RPC DI SUN

SUPPORTI



ARCHITETTURA



- 0) C trova S tramite PORTMAPPER
- 1) C invoca procedura remota \longrightarrow passa argomenti
- 2) C-stub costruisce richiesta ...
- 3) Passaggio nella rete
- 4) S-stub riceve/processa richiesta $\xrightarrow{\text{estrae procedura}}$ $\xrightarrow{\text{estrae argomenti}}$
- 5) S-stub fa richiesta locale a S
- 6) S elabora risultato procedura e passa a S-stub
- 7) S-stub costruisce risposta ...
- 8) Passaggio nella rete
- 9) C-stub riceve/processa risposta ...
- 10) C-stub passa risposta al C

CONTRATTO RPC

C'è infatti un contratto esplicito sulle info scambiate, diviso in 2 momenti:

- 1) CONTRATTO X SISTEMA DI NOMI \longrightarrow NUM. PROGRAMMA + NUM. VERSIONE + NUM. PROCEDURA
- 2) CONTRATTO X PROCEDURA REMOTA \longrightarrow 1 ARG. ENTRATA + 1 ARG. USCITA

Queste vengono realizzate con XDR

program NOMEPROG{
version NOMEVERS{
 return_type NOMEPROC(parm_type) = num_procedura ;
} = num_versione ;
} = num_prog_esadecimale ;

20000000h - 3FFFFFFF

PROCEDURE REMOTE

DATI PER ARGOMENTI

- \rightarrow bool \longrightarrow TRUE/FALSE \longrightarrow bool_t
- \rightarrow string \longrightarrow string nome <> \longrightarrow char* nome
- \rightarrow opaque \longrightarrow no tipo appartenenza
- \rightarrow void

Sviluppo REMOTO vs LOCALE

SERVER

- 1) Argomenti di ingresso/uscita passati per ARGOMENTO
- 2) Risultato deve puntare a VARIABILE STATICHE → allocazione globale, non si perde dopo procedura
- 3) Nome procedura cambia perché deve essere classificabile come da XDR
- 4) Nei parametri viene aggiunto struct svc_req* rqstp → contiene informazioni sulla richiesta remota

LE VARIABILI DI PROCEDURA VENGONO ALLOCATE NELLO STACK E SONO CANCELLATE QUANDO LA PROCEDURA TERMINA
COSÌ VIENE PASSATO CORRETTAMENTE A C-STUB

CLIENT

DEVE CONOSCERE → PROG + VERS + PROC
HOST REMOTO SERVIZIO

- 1) Abbiamo un GESTORE DI TRASPORTO client che viene richiesto al PORTMAPPER

CLIENT *clnt_create(char *host, u_long n_prog, u_long n_ver, char *proto);

- 2) Viene fatta invocazione remota con PUNTATORE A PARAMETRO + GESTORE DI TRASPORTO → NOME PROCEDURA CAMBIA
- Restituisce NULL o PUNTATORE A RISPOSTA
GESTIONE ERRORE
clnt_error()
SCELTA DI EFFICIENZA
passare il valore implica memoria, passare il riferimento no
- COPIATO DALLO STUB
- STUSSA MACCHINA C: C ← RP → STUB-C ← VALORE → STUSSA MACCHINA S: STUB-S ← RP → S
- MARSHALLING/UNMARSHALLING

PASSAGGI SVILUPPO

- 1) DEFINIRE SERVIZI + TIPI DATO → file.x
- 2) GENERAZIONE 2 STUB → rpcgen file.x
- 3) REALIZZARE CODICE CLIENT E SERVER
- 4) PUBBLICARE SERVIZI → PORTMAPPER
- 5) REPERIRE SERVER TRAMITE PORTMAPPER → clnt_create()

STRUTTURA MESSAGGIO RPC

```
struct rpc_msg {
    /* OGNI INVOCAZIONE E' UNIVOCATA */
    unsigned int xid;

    union switch (msg_type mtype) {
        case CALL: call_body cbody;
        case REPLY: reply_body rbody;
    } body;
};
```

```
enum msg_type { CALL = 0, REPLY = 1 };

enum reply_stat {MSG_ACCEPTED = 0, MSG_DENIED = 1};

enum accept_stat {
    SUCCESS=0, /* RPC execute with success */
    PROG_UNAVAIL = 1, /* remote hasn't exported program */
    PROG_MISMATCH = 2, /* remote can't support version # */
    PROC_UNAVAIL = 3, /* program can't support procedure */
    GARBAGE_ARGS = 4 /* procedure can't decode params */
};

enum reject_stat {
    RPC_MISMATCH = 0, /* RPC version number != 2 */
    AUTH_ERROR = 1 /* remote can't authenticate caller */
};
```

NON C'E' ESPlicitAMENTE MA CONTIENE

```
union reply_body switch (reply_stat stat) {
    case MSG_ACCEPTED: accepted_reply areply;
    case MSG_DENIED: rejected_reply rreply;
} reply;
```

```
struct call_body {
    unsigned int rpcvers;
    unsigned int prog;
    unsigned int vers;
    unsigned int proc; → CHIAMATA REMOTA
    opaque_auth cred;
    opaque_auth verf;
    /* procedure specific parameters start here */
};
```

SERVIZI RPC A LIVELLI

ALTO → livello utente

rnusers() → num utenti in nodo

rusers() → info utenti in nodo

rstat() → dati prestazioni verso nodo

rwall() → messaggio al nodo

getmaster() → nome nodo master

getrpeport() → porta dove c'è RPC

MEDIO → livello standard

callrpc(id)

↳ client provoca esecuzione di chiamata remota

registerrpc(id , procedura)

↳ nel registry viene registrata la procedura remota con la tupla

BASSO → livello configurazione

CLIENT

clntudp_create()

clnttcp_create()

clnt_call()

clnt_perror()

clnt_destroy()

SERVER

svcupd_create()

svctcp_create()

svc_register()

svc_run()

Server diventa un demone

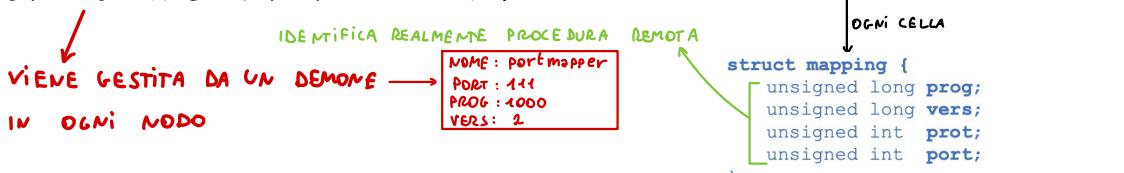
VISIONE SOLO TEORICA

TABELLA PORTMAPPER

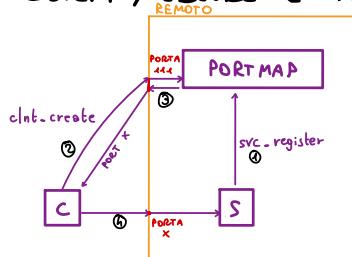
Si vuole associare ID a PROCEDURA ma l'implementazione è a lista dinamica.

LA TABELLA È UNA LISTA DINAMICA

```
struct *pmaplist { mapping map; pmaplist next; }
```



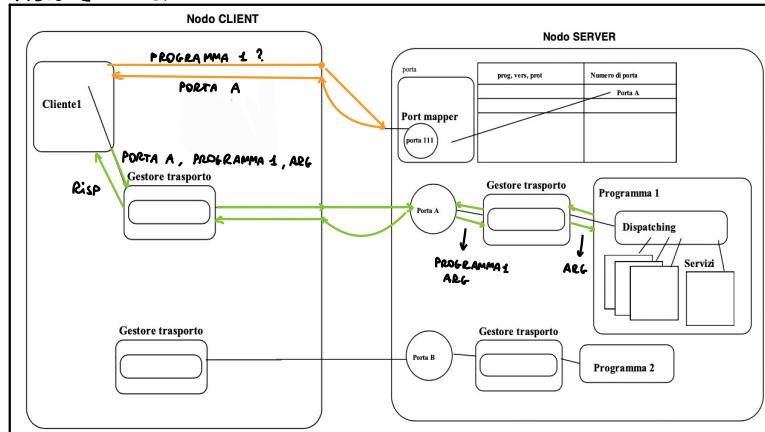
• CLIENT , SERVER E PORTMAPPER SI COORDINANO



Quo quello che succede è che

- 1) SERVER SI REGISTRA
- 2) CLIENTE CHIEDE A PORTMAPPER × SERVER
- 3) PORTMAPPER FORNISCE PORTA PER SERVER
- 4) CLIENTE CHIEDE SERVIZIO SERVER

VISIONE GENERALE



XDR

Formato comune di rappresentazione dei dati, ovvero ogni nodo possiede funzioni di conversione di questo formato → MINORE PERFORMANCE, MA SOLO 2*N FUNZIONI

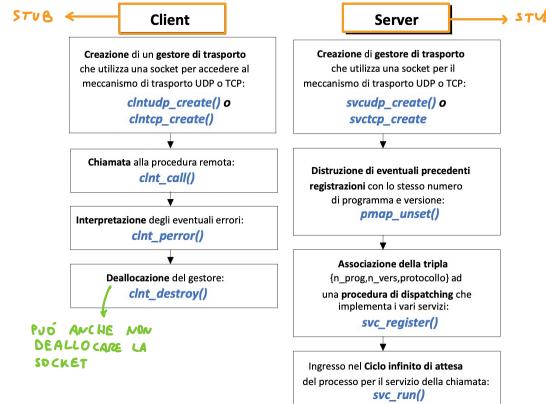
↪ ci sono funzioni come `xdr_int` o `xdr_short`

CHE VERIFICANO SE IL TIPO È CORRETTO E NEL

CASO LO SALVANO NELLA VERSIONE TRADOTTA

↪ QUANDO FACCIAMO RPCGEN, QUESTE FUNZIONI ATOMICHE VENGONO COMBINATE PER TRADURRE LE NOSTRE STRUTTURE DATO

UTILIZZO DI API DI BASSO LIVELLO



GESTORE DI TRASPORTO

SERVER

C'è struttura dati astratta `SVCXPRT`

`SVCXPTR* svcudp_create(int sock)`

`SVCXPTR* svctcp_create(int sock, u_int send_buff_size, u_int recv_buff_size)`

se `SOCK = RPC_ANYSOCK` → si crea socket

altrimenti si usa `sd = sock` (eventualmente si genera porta)

PUNTATORI ALLE OPERAZIONI SUI DATI

PUNTATO A 2 SOCKET → XP-SOCK → TRASPORTO SERVER

+ PORTA

XP_raddr → invio dati ESEC. REMOTA

CLIENT

C'è struttura dati CLIENT

`CLIENT* clntudp_create(sockaddr_in* addr, long progrnum, long versnum, timeval wait, int* sockp)`

`CLIENT* clnttcp_create(sockaddr_in* addr, long progrnum, long versnum, int* sockp, int sendsz, int recvsz)`

PER LE
RITRASMISSIONI

PROCEDURA di DISPATCHING (SERVER)

Dove sistemare i parametri, trovare procedura , invocarla, trattare risultato , consegnarlo ed exit.

```
bool svc_register(SVCXPRT* xptr, long progrnum, long versnum, char* dispatch, long protocol)
```

↳ ASSOCIA PROGRAMMA E VERSIONE ALLA PROCEDURA DI DISPATCHING

↳ PUNTATORE ALLA PROCEDURA DI DISPATCHING

```
void dispatch(svc_req* request, SVCXPRT* xprt)
```

↳ svc_getargs() —> ricavare parametri
↳ svc_sendreply() —> inviare risposta

CHIAMATA PROCEDURA REMOTA (CLIENT)

```
enum clnt_stat clnt_call(CLIENT* clnt, long procnum, xdrproc inproc, xdrproc outproc, char* in, char* out, timeval tout)
```

ROUTINE XDR
ARGOMENTO DA PASSARE
RISULTATO DA ASSEGNARE
 $NUM_RITRASMISS = TOUT/TIMOUT \leftarrow IN UDP \leftarrow$
timer per server irraggiungibile $\leftarrow IN TCP \leftarrow$

